

RU Staying

Software Engineering
01:332:452
Individual Contributions

By Group #11

Keya Patel
Zain Sayed
Mohammed Sapin
Purna Haque
Nga Man (Mandy) Cheng
Rameen Masood
Shilp Shah
Mathew Varghese
Thomas Tran
Eric Zhang

Github: <https://github.com/mohammedsapin/RUStaying>

1. program code writing—list explicitly which use cases or user interface screens were programmed by individual team members
(this list should correspond to the “product ownership” that was stated in the Customer Statement of Requirements in [Report #1](#))
2. [unit testing](#)[‡] (includes writing the program code to run unit tests and analyzing their coverage) and [integration testing](#)[‡] —breakdown by individual team members, applies to all remaining items
3. front-end design (HTML, CSS, JavaScript, [XML](#), if applicable)
4. integration and integration testing, cross-platform issues, if you are developing for desktop and smartphone[‡]
5. debugging
6. program documentation, including code comments, [technical documentation](#), [user documentation](#), etc...[‡]
7. **data collection (if applicable)[‡]**
 - writing scripts or programs for data collection (if applicable)[‡]
 - Other: _____
8. designing database tables and maintaining the database (if applicable)
9. brochure/flyer preparation—informativeness and general appearance of the product brochure
10. slides preparation—informativeness and general appearance of the slides
11. project management, including
 - opening accounts on public websites for the project
 - organizing meetings
 - coordinating activities
 - system evaluation with external users (if applicable)
 - installing and maintaining the developer’s resources
 - ([IDE](#), relational database, Web server, public archive, [version control](#), ...)combining all of the contributions into the correct files and formats
 - Other: _____ (include everything that you believe contributed towards making your demo successful)

Main Individual Contribution Table

Keya	10%	10%	10%	10%	10%	10%	10%
Zain	10%	10%	10%	10%	10%	10%	10%
Mohammed	10%	10%	10%	10%	10%	10%	10%
Purna	10%	10%	10%	10%	10%	10%	10%
Nga Man	10%	10%	10%	10%	10%	10%	10%
Rameen	10%	10%	10%	10%	10%	10%	10%
Shilp	10%	10%	10%	10%	10%	10%	10%
Mathew	10%	10%	10%	10%	10%	10%	10%
Eric	10%	10%	10%	10%	10%	10%	10%
Thomas	10%	10%	10%	10%	10%	10%	10%

Code Writing / Unit Testing

Guest Services-Eric Zhang, Rameen Masood, Thomas Tran

In the first individual contributions document for demo 1, we described our process of creating our bellboy service. For demo 2, we added maintenance, travel and valet, and room service guest requests, in addition to updating bellboy. As of demo 2, we had the following information on each page:

Bellboy:

- Select a date to carry out request (current or future date)
- Select a time to carry out request
- Input text box to describe where to pick up luggage
- Select a value for number of bags guest wants to be picked up

Travel and Valet:

- Select a date to carry out request (current or future date)
- Select a time to carry out request
- Enter a starting and destination address

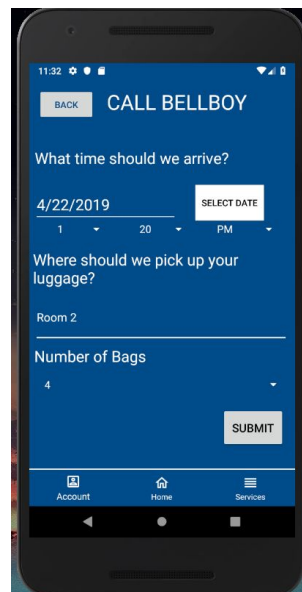
Maintenance:

- Select a date to carry out request (current or future date)
- Select a time to carry out request
- Select bathroom, electronic, and lighting issues
- Input text box with a short description of the issues

Room Service:

- Select a date to carry out request (current or future date)
- Select a time to carry out request
- Select restocking of towels, bedsheets, soap, and cleaning service
- Input text box with short description of request (number of towels needed, etc.)

Below is an example of our new filled out bellboy request page:



The way we implemented these services were similar- for the time of request, we used spinners, for the date selection we used a calendar widget with local time stored, for input boxes and checkboxes we used their corresponding widgets in android studios. To store and pass the data to our database, we use a similar method as we did for bellboy service in demo 1- in our Service.java class, we used overloading to create 4 different service objects with a different number of parameters for each, each with their own getter and setter methods. See below:

```
Service.java
74 //Travel and Valet Object
75 public Service(String requestType, String requestedTimeValet, String requestDate, String answer1, String answer2,
76               String answer3, String answer4) {
77     this.requestType = requestType;
78     this.requestedTimeValet = requestedTimeValet;
79     this.requestDate = requestDate;
80     this.startingStreet = answer1;
81     this.destinationStreet = answer2;
82     this.startingCityStateZip = answer3;
83     this.destinationCityStateZip = answer4;
84 }
85
86 //Maintenance Object
87 public Service(String requestType, String requestDate, String requestedTimeMaintenance, String inputs, String bathroom,
88               String electronic, String lighting, String checkboxes) {
89     this.requestType = requestType;
90     this.requestDate = requestDate;
91     this.inputs = inputs;
92     this.bathroom = bathroom;
93     this.electronic = electronic;
94     this.lighting = lighting;
95     this.checkboxes = checkboxes;
96     this.requestedTimeMaintenance = requestedTimeMaintenance;
97 }
98
99 //Room Service Object
100 public Service(String requestType, String requestDate, String requestedTimeRoomService, String inputs,
101               String towels, String soap, String bedsheets, String cleaningservice, String checkboxes) {
102     this.requestType = requestType;
103     this.requestDate = requestDate;
104     this.inputs = inputs;
105     this.checkboxes = checkboxes;
106     this.towels = towels;
107     this.soap = soap;
108     this.bedsheets = bedsheets;
109     this.cleaningservice = cleaningservice;
110     this.requestedTimeRoomService = requestedTimeRoomService;
```

As a result, for example in our bellboy request page, once the use had filled out their fields in the form, we would use these attributes to create a service object with these parameters, and the service class would automatically classify this object as a bellboy object. Passing these objects in as JSON objects to google firebase, we insert the object into the Service branch, then into the user ID child branch, then create the request information with its randomly generated ID as children of the user ID branch. We can see an example of a screenshot of our database which is updated in real time:



For our final submission, we are also planning on creating a new service request for food room service, for which we will create a menu with food options and pricing and the ability to order food for a certain time to a specific place in the hotel, with the overall backend process being the same as the 4 other services.

Reservation Activity - Shilp Shah, Mathew Varghese

Once of the main conveniences of the app is to be able to make reservations very easily through our app. After updating the XML files to include CardView along with recyclerView, we started working on the functionality of the feature. This was a little more difficult than expected and involved a lot of data manipulation and sending data through multiple activities.

The first step was to see what information would be needed to display the rooms that could be reserved. So made a screen where one could select a check-in date and check out date for their stay. They also could filter certain rooms based on type. The options that we had listed were single, double, queen, and king. After they had filled out this information they could click on the view available rooms button, which would take them to the recyclerView. The code below shows how we send the corresponding data to other activities.

```
if(checkInDate != null && checkOutDate != null)
{
    //Check checkIn and checkOut dates to make sure they are not null

    //Setup ResInfo object
    info = new ResInfo(checkInDate, checkOutDate, roomTypes);

    Intent viewRooms = new Intent( packageContext: ReservationActivity.this, newViewRooms.class);

    Bundle b = new Bundle();

    b.putInt("inDay", info.getCheckIn().getDayOfMonth());
    b.putInt("inMonth", info.getCheckIn().getMonth().getValue());
    b.putInt("inYear", info.getCheckIn().getYear());

    b.putInt("outDay", info.getCheckOut().getDayOfMonth());
    b.putInt("outMonth", info.getCheckOut().getMonth().getValue());
    b.putInt("outYear", info.getCheckOut().getYear());

    b.putString("checkIn", info.getCheckIn().toString());
    b.putString("checkOut", info.getCheckOut().toString());
    b.putStringArray("roomTypes", info.getRoomTypes());

    viewRooms.putExtra( name: "resInfo", b);

    startActivity(viewRooms);
}
```

We also needed a way to determine if a room was booked for the the days that the current user is interested in order to fill the booked rooms out appropriately. Therefore, we created a new object to hold all the corresponding reservation called ResInfo. It would hold the information depicted below. By comparing check in and check out dates of the reservation object, we were able to determine a if there is already a booking for a room during the dates of interest.

```

package com.example.rustaying;
import java.time.LocalDate;






public class ResInfo {
    private LocalDate checkIn;
    private LocalDate checkOut;
    private String [] roomTypes = new String[4]; //Hold types of rooms guest wants

    //Constructors
    public ResInfo(){

    }
    public ResInfo(LocalDate checkIn, LocalDate checkOut, String[] roomTypes)
    {
        this.checkIn = checkIn;
        this.checkOut = checkOut;
        for(int i = 0; i < roomTypes.length; i++)
        {
            this.roomTypes[i] = roomTypes[i];
        }
    }
}

```

The recyclerView provided a real-time visualization of all the rooms that were available to be booked. It does this by having a list of of cardViews that you can scroll through. Each ‘Card’ has a small thumbnail where the image of the room could be inserted. It also displays the type of room, a description for the room, the room number, and it’s pricing.

	Single 01 \$250	BOOK
	Single 02 \$250	BOOK
	Single 03 \$250	BOOK
	Single 04 \$250	BOOK
	Single 05 \$250	BOOK
	Double 6 \$300	BOOK

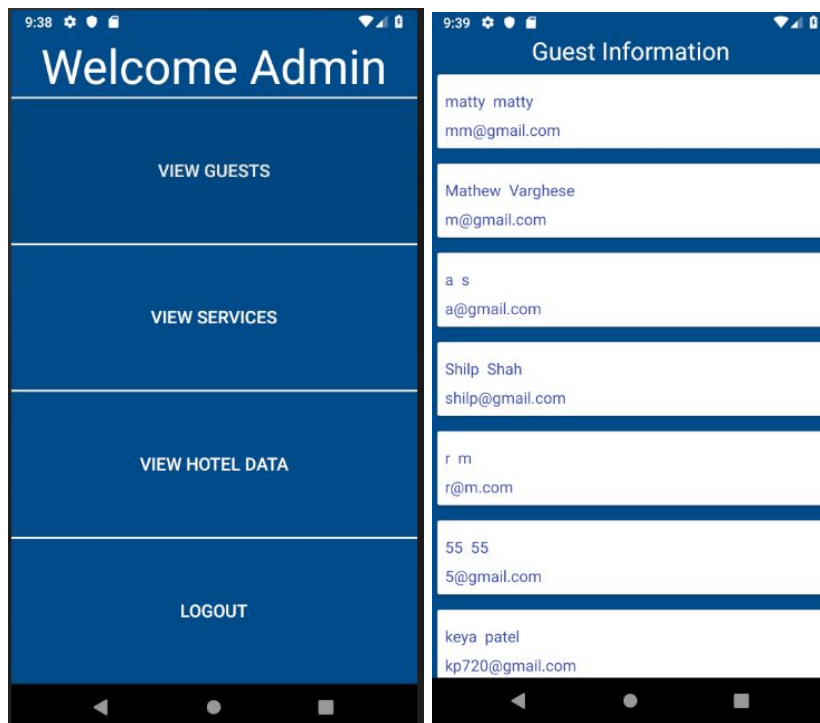
We ran into some challenges when it came to combining the Cards into the recyclerView. But ultimately, this was just a matter of growing more comfortable with android studio's tools, and after some time of playing around with the code, we managed to get everything up and running. We started with our implementation of the adapter class, that would prepare the display data in a way that the recycler would then be able to accept. We setup all the text, image, and button information for a card using our implementation of the viewHolder class, which is a class defined within our adapter class. We call viewHolder methods to set these values elsewhere within methods of the adapter class. After the adapter sets up the necessary objects and variables, It is passed to the viewRooms class which then populates the recyclerView with Cards.

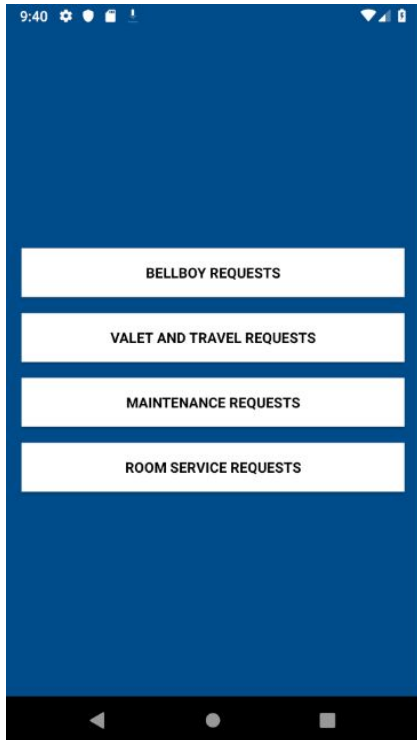
Administration/staff pages (Mohammed Sapin, Purna Haque, Nga Man Cheng)

The administration page was updated from Demo 1 to have multiple staff accounts. The admin page can view requests for each category Bellboy, Valet & Travel, Maintenance, and Room Service. The individual pages have similar functionalities just with different parameters. For example bellboy requests all have parameters like amount of luggage, pickup location, time and date; however valet and travel requests will have different parameters like pick up and drop off location, time, and date, etc.

Admin Page

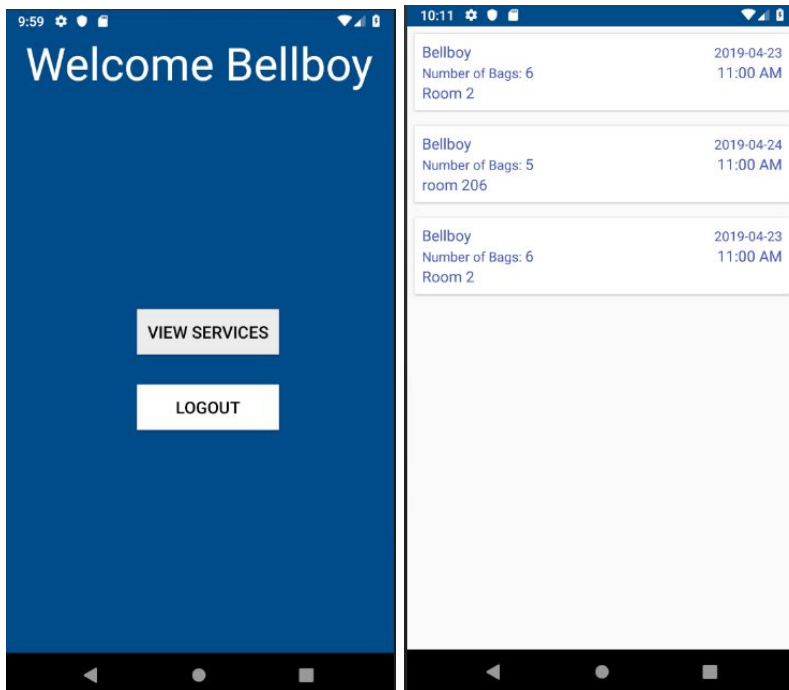
The admin page is only accessible to administrators with the correct email and password login. It allows the admin to view hotel guest's information and all the service requests made by the guests. Each service is filtered into different categories of specific requests made (bellboy, valet & travel, maintenance, and room service).





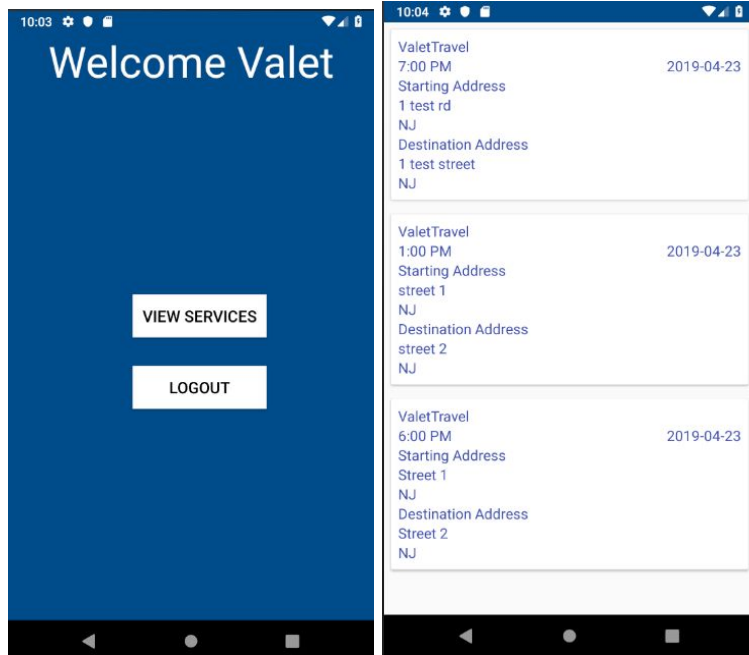
Bellboy Page

The bellboy admin page is only accessible to bellboys with the correct email and password login. They will see the real time requests that guests inputs and will know which requests to handle first.



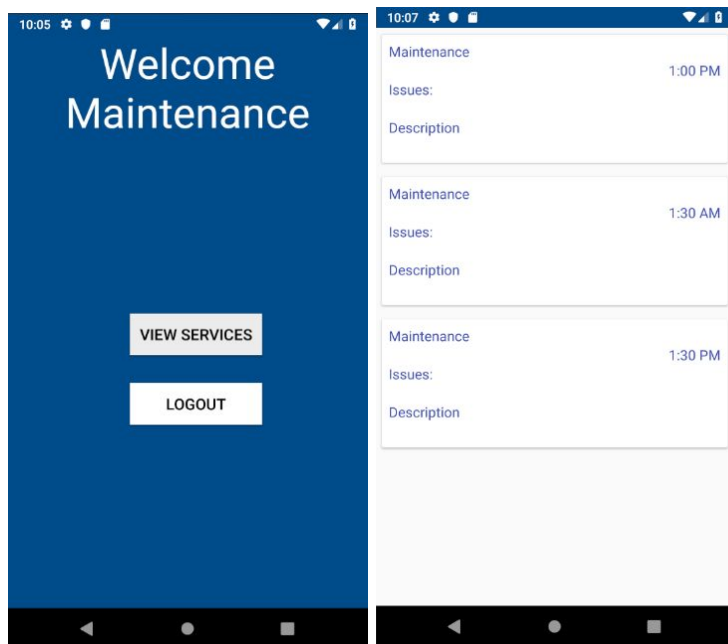
Valet & Travel Page

The valet and travel admin page is only accessible to valet with the correct email and password login. They will see the real time requests that guests inputs and will know which requests to handle first.



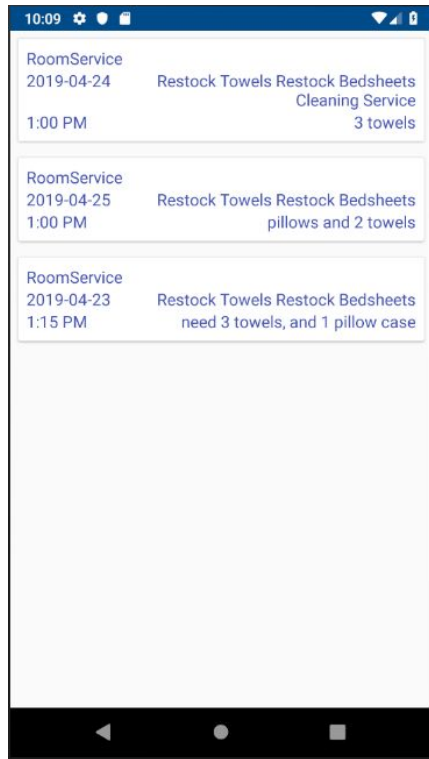
Maintenance Page

The maintenance admin page is only accessible to maintenance staff with the correct email and password login. They will see the real time requests that guests inputs and will know which requests to handle first.



Room Service Page

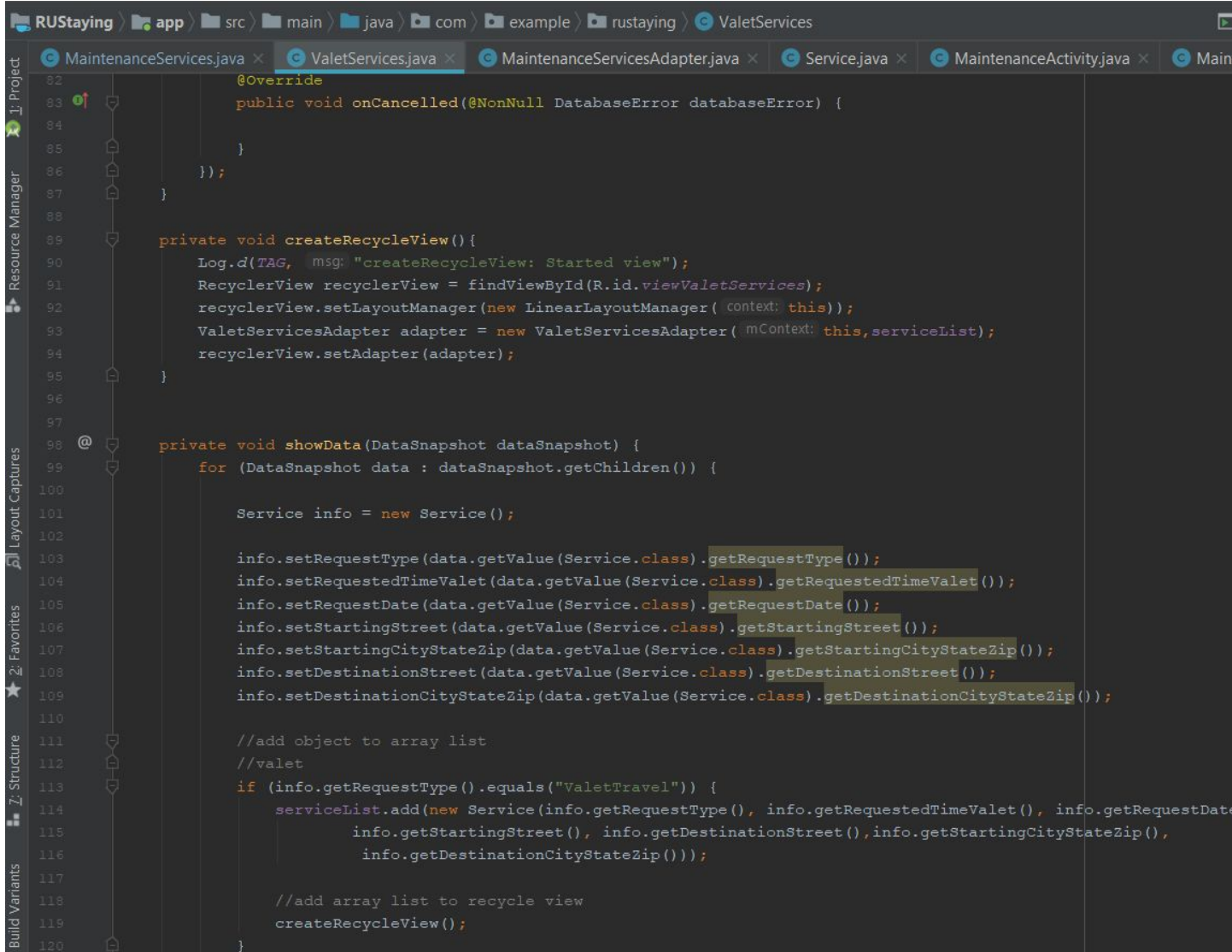
The room service admin page is only accessible to room service staff with the correct email and password login. They will see the real time requests that guests inputs and will know which requests to handle first.



RoomService	Restock Towels Restock Bedsheets
2019-04-24	Cleaning Service
1:00 PM	3 towels
RoomService	Restock Towels Restock Bedsheets
2019-04-25	pillows and 2 towels
1:00 PM	
RoomService	Restock Towels Restock Bedsheets
2019-04-23	need 3 towels, and 1 pillow case
1:15 PM	

ViewService.java

This is just a small snippet to show how was gather all the requests that is entered in real time.
We filter all the requests and show them to specific users/admins based on the login page.



```
RUStaying > app > src > main > java > com > example > rustaying > ValetServices
MaintenanceServices.java x ValetServices.java x MaintenanceServicesAdapter.java x Service.java x MaintenanceActivity.java x Main
82 @Override
83 public void onCancelled(@NonNull DatabaseError databaseError) {
84
85 }
86
87 });
88
89 private void createRecyclerView() {
90     Log.d(TAG, "msg: \"createRecyclerView: Started view\"");
91     RecyclerView recyclerView = findViewById(R.id.viewValetServices);
92     recyclerView.setLayoutManager(new LinearLayoutManager(context, this));
93     ValetServicesAdapter adapter = new ValetServicesAdapter(mContext, this, serviceList);
94     recyclerView.setAdapter(adapter);
95 }
96
97
98 @
99 private void showData(DataSnapshot dataSnapshot) {
100     for (DataSnapshot data : dataSnapshot.getChildren()) {
101
102         Service info = new Service();
103
104         info.setRequestType(data.getValue(Service.class).getRequestType());
105         info.setRequestedTimeValet(data.getValue(Service.class).getRequestedTimeValet());
106         info.setRequestDate(data.getValue(Service.class).getRequestDate());
107         info.setStartingStreet(data.getValue(Service.class).getStartingStreet());
108         info.setStartingCityStateZip(data.getValue(Service.class).getStartingCityStateZip());
109         info.setDestinationStreet(data.getValue(Service.class).getDestinationStreet());
110         info.setDestinationCityStateZip(data.getValue(Service.class).getDestinationCityStateZip());
111
112         //add object to array list
113         //valet
114         if (info.getRequestType().equals("ValetTravel")) {
115             serviceList.add(new Service(info.getRequestType(), info.getRequestedTimeValet(), info.getRequestDate(),
116                 info.getStartingStreet(), info.getDestinationStreet(), info.getStartingCityStateZip(),
117                 info.getDestinationCityStateZip()));
118
119             //add array list to recycle view
120             createRecyclerView();
121         }
122     }
123 }
```

Profile Activity / Feedback - Zain Sayed, Keya Patel

One aspect of the code we were missing after the first demo was the ability to view the profile of the user currently logged in. The main task was to figure this out and along the way I decided to add a way to edit the users information if they so choose to. Since we had done had pulled data from the JSON tree before the implementation was relatively standard. Reusability of past code was a big plus. The hard part was to update specific fields of the user if they want to update.

Using the Guest objects getter and setter methods, I was able to pull the data in a form of strings and set their value into the TextView box so that it was easier to display said value. ProfileActivity.java below.

```
Log.d(TAG, msg: "onAuthStateChanged:signed in");
myRef.child("Guest").child(userID).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        showData(dataSnapshot);

        TextView fname2 = (TextView) findViewById(R.id.fname2);
        String firstName = g.getFirstName();
        fname2.setText(firstName);

        TextView lname2 = (TextView) findViewById(R.id.lname2);
        String lastName = g.getLastName();
        lname2.setText(lastName);

        TextView email2 = (TextView) findViewById(R.id.email2);
        String email = g.getGuestEmail();
        email2.setText(email);
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
}
```

The TextView boxes as seen below automatically update once the user clicks on the View Profile part on the navigation bar present on all pages.(Side note: Profile Information is instantaneously updated once its been edited). An edit info button was added so that the user has good accessibility to edit info and takes no longer than 2 redirects to get to the edit page.

The image displays two side-by-side screenshots of a mobile application interface, specifically the 'Guest Information' screen. Both screens have a solid blue background. The left screenshot shows the initial state where the labels 'First Name', 'Last Name', and 'Email' are positioned above empty 'TextView' boxes. The right screenshot shows the same screen with the input boxes highlighted by dashed blue borders, indicating they are active or being edited. Both screens feature a white 'EDIT INFO?' button at the bottom right. At the very bottom, there is a navigation bar with three icons: a person icon for 'Account', a house icon for 'Home', and a square icon for 'Services'.

The edit info page on the other hand was a little bit more difficult to implement since there was no precedence of updating info on our demo1 version of the project. The challenge was to update individual fields of the JSON Guest object even if all the fields of input are not filled out/empty. Seen below is the XML which takes the input.

11:01

Edit your info

Please enter your first name

First name here

Please enter your last name

Last name here

Please enter your new email address

Email here

SUBMIT

Account Home Services

After reading these inputs, the values are inserted into a hashmap which then are uploaded directly to the database. Firebase has the capability of reading these hashmaps and inserting them correctly in the right place based on the userid of the Guest object db table. The code below does the uploading.

```
final String answer1 = first_name.getText().toString().trim();
final String answer2 = last_name.getText().toString().trim();
final String answer3 = emailID.getText().toString().trim();

Map<String, Object> list = new HashMap<>();

if(!answer1.isEmpty()) {
    list.put("firstName", answer1);
}

if(!answer2.isEmpty()) {
    list.put("lastName", answer2);
}

if(!answer3.isEmpty()) {
    list.put("guestEmail", answer3);
}
```

Forgot Password -Zain Sayed

One functionality which was also needed after demo one was the ability for the user to retrieve the account associated with the email they first registered with. The new functionality now was able to bring the user to predefined page where the user would input their email. If it wasn't a valid one an error would show up and if it was Firebase has the capability of sending a reset email to the user's email through which it was possible to recover the account

The image displays two side-by-side mobile application screens with a dark blue background and white text.

The left screen is the login page. At the top, it features a crown icon and the text "RUSTAYING" and "VARGHESE HOTEL". Below this, there are two input fields: "Email" and "Password", each with a wavy underline. A white "LOGIN" button is positioned below the password field. At the bottom of the screen, there is a "Forgot Password?" link.

The right screen is titled "Reset Password". It has a single "Email" input field with a wavy underline. Below the input field is a grey button labeled "SEND EMAIL". At the bottom of this screen, there is a navigation bar with three icons and labels: "Account" (person icon), "Home" (house icon), and "Services" (square icon).

```
firebaseAuth.sendPasswordResetEmail(userEmail).addOnCompleteListener((t
```

This above function takes the email and adds a listener to it, once user hits send email, the Firebase host will take control and complete the request.

For the feedback activity, we took the comments from demo 1 into consideration when redesigning the feature. Below is the new UI and updates questions for the guests to answer in the feedback.

The image shows two mobile app screens for a feedback form. The left screen has a dark blue background and contains the following text and elements:

- "If you used our Room Service feature, please rate it:" followed by five stars.
- "If you used our Bellboy feature, please rate it:" followed by five stars.
- "If you used the Valet Service feature, please" followed by five stars.
- "If you used the Maintenance Service" followed by five stars.
- "Any additional comments or concerns:" followed by a text input field labeled "Type here" and a "SUBMIT" button.

The right screen has a lighter blue background and contains the following text and elements:

- "Feedback Form" title.
- "Please rate your experience" followed by five stars.
- "Which of these features did you use:" followed by four checkboxes: "Room Service", "Bellboy Services", "Valet Services", and "Maintenance".
- "If you used our Online Check-In feature, did you find it convenient?" followed by "No" and "Yes" radio buttons.
- "If you used our Room Service feature, please rate it:" followed by five stars.

Both screens have a bottom navigation bar with three icons: "Account", "Home", and "Services".

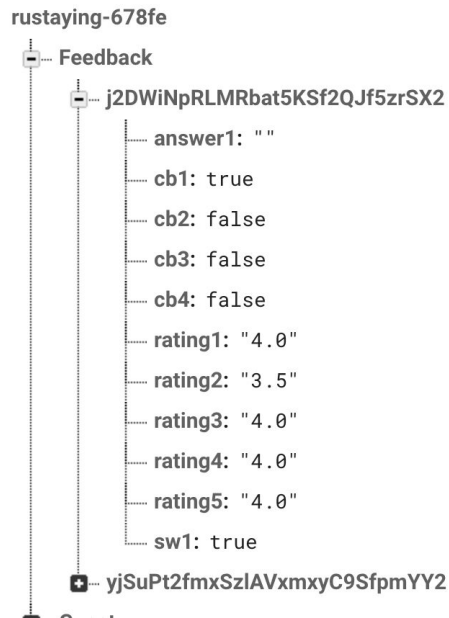
Now all the questions are numerical based so we can do simple analysis on it. The data is stored in the feedback under the `userId`, then on the admin side, the data will be pulled to create graphs. Additionally, another issue we overlooked was the guest should not be able to submit feedback if they are not checked into the hotel so we added error checking to only give access to guests that are checked in.

```
feedbackBtn = (Button)findViewById(R.id.feedbackBtn);
String userID = user.getId();

myRef.child("Guest").child(userID).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        showData(dataSnapshot);
        final Boolean check = g.isCheckedIn();

        feedbackBtn.setOnClickListener(v -> {
            if (check){
                Intent feedback = new Intent( packageContext: HomeActivity.this, FeedbackActivity.class);
                startActivity(feedback); //Redirect to feedback page
            }
            else{
                Toast.makeText( context: HomeActivity.this, text: "You don't have access to Feedback since you are not checked-in.", Toast.LENGTH_SHORT).show();
            }
        });
    }
});
```

This is how the data is stored in firebase and it will be used by the admin in a later feature.



Each entry is located under the userID so if a user submits multiple feedback forms for the same reservation, then we will only use the most recent submission. However, if the user has multiple reservations, then there will be two separate feedback entries under that user.